

APPENDIX A

```

1  package ihn.dialex;

2  import ihn.merkato.*;
3  import ihn.math.Gaussian;
4  import ihn.math.Normal;
5  import java.util.Hashtable;
6  import java.util.Random;

7  /**
8   * @version $Revision: 1.6 $, $Date: 1998/12/22 05:21:22 $
9   * @author Nemo Semret
10  */

11  public class ReservationCalculator extends Hashtable {
12      int V2=1;
13      int C,B;
14      float lambda, mu,r;
15      int dt;
16      int horizon;

17      public ReservationCalculator(int C,int B,float mu, float lambda,
18                                  float r, int horizon, int dt) {
19          this.C = C;
20          this.B=B;
21          this.mu=mu;
22          this.lambda=lambda;
23          this.r=r;
24          this.horizon= horizon;
25          this.dt = dt;
26          System.out.println(this);
27      }

```

```

28     public float interArrival() {
29         return (float) (1.0/lambda);
30     }

31     public String toString() {
32         return super.toString()+" C = "+C +" B="+B +" mu="+mu +
33             " lambda="+lambda+" r="+r+" horiz="+horizon+" dt="+dt;
34     }

35     float A, K3, D, K4;
36     /**
37      *Compute beta, K1, K2 for each m
38      *and  A, K3, D, K4.
39      */
40     public void init() throws ArithmeticException {

41         float [] K1;
42         float [] K2;
43         float [] beta;

44         K1 = new float [B+1];
45         K2 = new float [B+1];
46         beta = new float [B+1];
47         float [] w0 = new float [B+1];
48         float [] w1 = new float [B+1];
49         Gaussian phi = new Gaussian(0,1);
50         float root2overpi = (float)Math.sqrt(2.0/Math.PI);
51         float rho = lambda/mu;
52         float zob = rho/(1+rho);
53         float plop = 1/(1+rho);

54         A= K3= D= K4=0;

```

```

55     float rC = r*C;
56     float rootrC = (float)Math.sqrt(rC);
57     float u;

58

59     for (int m=0 ; m<=B; m++, plop=plop*zob) {
60         if(m>=4*V2) {
61             u = (1- (float)Math.sqrt(1-(float)(4*V2)/m))/2;
62             w0[m]=(float)Math.sqrt(u*m/(1-u));
63             w1[m]=(float)Math.sqrt((1-u)*m/u);
64             K1[m]= 2*phi.integral(w0[m],w1[m],(float)1.0);

65             K2[m]= root2overpi*((float)Math.exp(-w0[m]*w0[m]/2)*(w0[m]/m - (float) 1.0/w0[m]) -
66                 (float)Math.exp(-w1[m]*w1[m]/2)*(w1[m]/m - (float) 1.0/w1[m]))+2*K1[m];

67             if(m<B) beta[m]=plop;
68             else beta[m]= plop*rho;

69             A -=beta[m]*rC/m;
70             K3 +=beta[m]*rC*((m-1)*K1[m]/m + K2[m]/m);
71             D+= beta[m]*rootrC/m;
72             K4 -=beta [m]*K1[m]*rootrC;
73         }
74     }

75     System.out.println("A="+A+" K3="+K3+" D="+D+" K4="+K4);
76 }

77 /**
78  * Generates a forward sample path of the process P.
79  * @param p0 The initial price P[0]=po;
80  * @param T The duration.

```

```

81      * @return An array of size 1+floor(T/dt)
82      */
83      float [] fwdSamplePath(float p0, int T, int dt ) {
84          float dP=0, dW=0;
85          float []P = new float[1+T/dt];
86          // Normal norm = new Normal(100);
87          Random rand = new Random(System.currentTimeMillis());
88          P[0]=p0;
89          int i,t;

90          for( t=dt, i=1; t<=T; i++, t+=dt) {
91              // dP = A*P[i-1]+K3*dt + (D*P[i-1]+K4)*norm.nextSample()*(float)Math.sqrt(dt);
92              dP = A*P[i-1]+K3*dt + (D*P[i-1]+K4)
93                  *(float) (rand.nextDouble() < 0.5 ? -1.0 : 1.0)
94                  *(float) Math.sqrt(dt);
95              P[i]=P[i-1]+ dP;
96          }

97          return P;
98      }

99      /**
100      * For each time t between the current and T, we get an array
101      * which contains the histogram of P[t].
102      * @param NBINS Number of bins in the histogram.
103      * @param p0 The current price.
104      *
105      */
106      int [][] monteCarlo(float p0, int T) {
107
108          int [][] histog = new int[1+T/dt][NBINS];
109          float [] samplepath ;

```

```

110     int n,t,i;
111     int p;

112     for ( n=0; n<nsamples; n++) {
113         samplepath = fwdSamplePath(p0,T,dt);
114         for (t=0, i=0; t<=T; i++, t+=dt) {
115             p=(int)( samplepath[i]*NBINS);
116             p = p<0 ? 0 :p;
117             p= p>NBINS-1? NBINS-1 :p;
118             histog[i][p]++;
119         }
120     }

121     return histog;
122 }

123 /**
124  * Number of bins in each histogram
125  */
126 int NBINS=10;
127 /**
128  * Number of sample paths for each fastFwd
129  */
130 int nsamples=100;

131 public synchronized void fastFwd(float p0) {
132     if( get(new Integer((int)(p0*NBINS))) == null) {
133         System.out.print("FFing "+horizon+" for price "+p0);
134         long a=System.currentTimeMillis();
135         put(new Integer((int)(p0*NBINS)), monteCarlo(p0, horizon));
136         System.out.println(" done in "+(System.currentTimeMillis()-a));
137     }

```

```

138     }

139     /**
140     * @param p0 The current market price.
141     * @param b The reservation for which the fee is computed.
142     */
143     public Reservation fee(Bid b, float p0) {
144         int x,t,i;
145         float fee=0;
146         float duration = Math.min(b.qty,horizon);
147         int [][] histogram = (int [][]) get(new Integer((int)(p0*NBINS)));

148         if(histogram == null) {
149             fastFwd(p0);
150             histogram = (int [][]) get(new Integer((int)(p0*NBINS)));
151         }

152         for(i=0, t=0; t<=duration; t+=dt, i++)
153             for( x=0; x<NBINS; x++)
154                 fee += (Math.max((x+0.5)/((float)NBINS) - b.price,0)
155                     * histogram[i][x])/nsamples;

156         fee = fee*dt;

157         Reservation foo=new Reservation(b);
158         b.qty =duration;
159         foo.fee= fee;
160         // System.out.println(p0+" "+foo.price+" "+foo.qty+" "+foo.fee+" "+foo.bidderid);
161         return foo;
162     }

163 }

```